

# Foundations of Artificial Intelligence

## 4. Informed Search Methods

Heuristics, Local Search Methods, Genetic Algorithms

Joschka Boedecker and Wolfram Burgard and  
Frank Hutter and Bernhard Nebel and Michael Tangermann



Albert-Ludwigs-Universität Freiburg

May 8, 2019

- 1 Best-First Search
- 2 A\* and IDA\*
- 3 Local Search Methods
- 4 Genetic Algorithms

- 1 Best-First Search
- 2 A\* and IDA\*
- 3 Local Search Methods
- 4 Genetic Algorithms

# Best-First Search

Search procedures differ in the way they determine the next node to expand.

**Uninformed Search:** Rigid procedure with no knowledge of the cost of a given node to the goal.

**Informed Search:** Knowledge of the worth of expanding a node  $n$  is given in the form of an evaluation function  $f(n)$ , which assigns a real number to each node. Mostly,  $f(n)$  includes as a component a heuristic function  $h(n)$ , which estimates the costs of the cheapest path from  $n$  to the goal.

$$f(n) = g(n) + h(n)$$

**Best-First Search:** Informed search procedure that expands the node with the “best”  $f$ -value first.

**function** TREE-SEARCH(*problem*) **returns** a solution, or failure  
initialize the frontier using the initial state of *problem*

**loop do**

**if** the frontier is empty **then return** failure

→ choose a leaf node and remove it from the frontier

**if** the node contains a goal state **then return** the corresponding solution

expand the chosen node, adding the resulting nodes to the frontier

Best-first search is an instance of the general TREE-SEARCH algorithm in which *frontier* is a priority queue ordered by an evaluation function  $f$ .

When  $f$  is always correct, we do not need to search!

# Greedy Search

A possible way to judge the “worth” of a node is to estimate its path-costs to the goal.

$$h(n) = \text{estimated path-costs from } n \text{ to the goal}$$

The only real restriction is that  $h(n) = 0$  if  $n$  is a goal.

A best-first search using  $h(n)$  as the evaluation function, i.e.,  $f(n) = h(n)$  is called a *greedy search*.

Example: route-finding problem:

$$h(n) =$$

A possible way to judge the “worth” of a node is to estimate its path-costs to the goal.

$$h(n) = \textit{estimated path-costs from } n \textit{ to the goal}$$

The only real restriction is that  $h(n) = 0$  if  $n$  is a goal.

A best-first search using  $h(n)$  as the evaluation function, i.e.,  $f(n) = h(n)$  is called a *greedy search*.

Example: route-finding problem:

$h(n)$  = straight-line distance from  $n$  to the goal

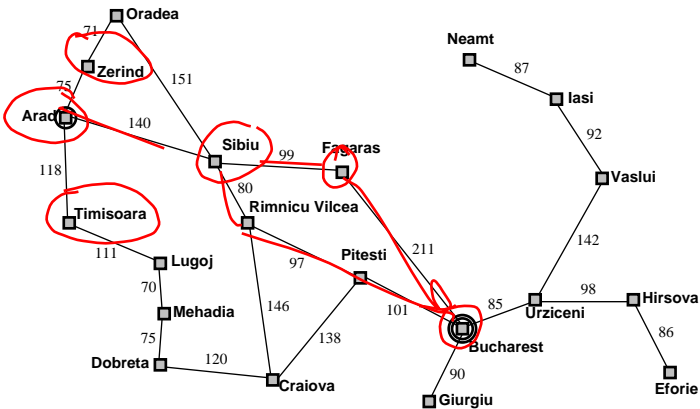
The evaluation function  $h$  in greedy searches is also called a heuristic function or simply a heuristic.

- The word *heuristic* is derived from the Greek word  $\epsilon\upsilon\rho\iota\sigma\kappa\epsilon\iota\upsilon\nu$  (note also:  $\epsilon\upsilon\rho\eta\kappa\alpha!$ )
- The mathematician Polya introduced the word in the context of problem solving techniques.
- In AI it has two meanings:
  - Heuristics are fast but in certain situations incomplete methods for problem-solving [Newell, Shaw, Simon 1963] (The greedy search is actually generally incomplete).
  - Heuristics are methods that improve the search in the average-case.

→ In all cases, the heuristic is *problem-specific* and *focuses* the search!



# Greedy Search Example



|                |     |
|----------------|-----|
| Arad           | 366 |
| Bucharest      | 0   |
| Craiova        | 160 |
| Drobeta        | 242 |
| Eforie         | 161 |
| Fagaras        | 176 |
| Giurgiu        | 77  |
| Hirsova        | 151 |
| Iasi           | 226 |
| Lugoj          | 244 |
| Mehadia        | 241 |
| Neamt          | 234 |
| Oradea         | 380 |
| Pitesti        | 100 |
| Rimnicu Vilcea | 193 |
| Sibiu          | 253 |
| Timisoara      | 329 |
| Urziceni       | 80  |
| Vaslui         | 199 |
| Zerind         | 374 |

# Greedy Search from Arad to Bucharest

(a) The initial state



# Greedy Search from Arad to Bucharest

(a) The initial state



(b) After expanding Arad



# Greedy Search from Arad to Bucharest

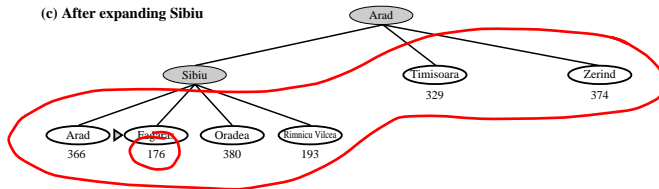
(a) The initial state



(b) After expanding Arad



(c) After expanding Sibiu



# Greedy Search from Arad to Bucharest

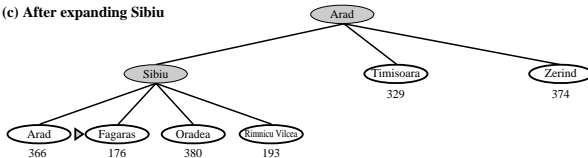
(a) The initial state



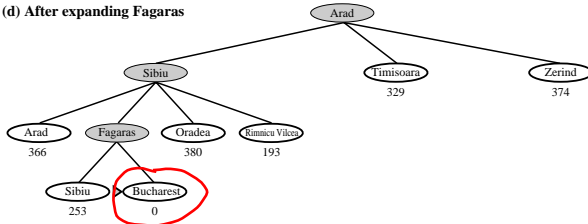
(b) After expanding Arad



(c) After expanding Sibiu



(d) After expanding Fagaras



# Greedy Search - Properties

- a good heuristic might reduce search time drastically
- non-optimal
- incomplete
- graph-search version is complete only in finite spaces

Can we do better?

# Lecture Overview

- 1 Best-First Search
- 2 **A\* and IDA\***
- 3 Local Search Methods
- 4 Genetic Algorithms

# A\*: Minimization of the Estimated Path Costs

A\* combines greedy search with the uniform-cost search:

Always expand node with lowest  $f(n)$  first, where

$g(n)$  = actual cost from the initial state to  $n$ .

$h(n)$  = estimated cost from  $n$  to the nearest goal.

$$f(n) = g(n) + h(n),$$

the estimated cost of the cheapest solution through  $n$ .

Let  $h^*(n)$  be the actual cost of the optimal path from  $n$  to the nearest goal.  $h$  is admissible if the following holds for all  $n$ :

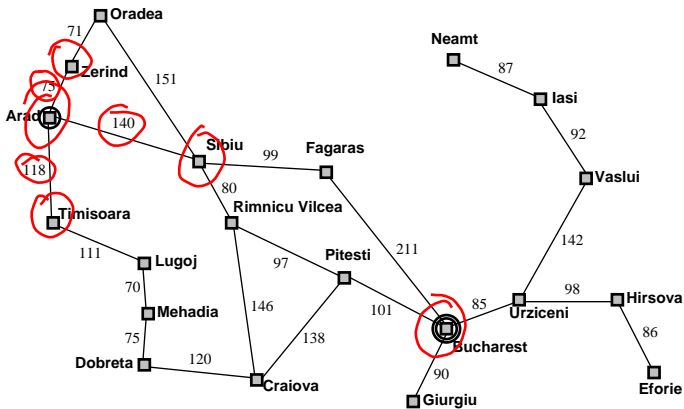
$$h(n) \leq h^*(n)$$

We require that for A\*,  $h$  is admissible (example: straight-line distance is admissible).

In other words,  $h$  is an optimistic estimate of the costs that actually occur.



# A\* Search Example



|                |     |
|----------------|-----|
| Arad           | 366 |
| Bucharest      | 0   |
| Craiova        | 160 |
| Drobeta        | 242 |
| Eforie         | 161 |
| Fagaras        | 176 |
| Giurgiu        | 77  |
| Hirsova        | 151 |
| Iasi           | 226 |
| Lugoj          | 244 |
| Mehadia        | 241 |
| Neamt          | 234 |
| Oradea         | 380 |
| Pitesti        | 100 |
| Rimnicu Vilcea | 193 |
| Sibiu          | 253 |
| Timisoara      | 329 |
| Urziceni       | 80  |
| Vaslui         | 199 |
| Zerind         | 374 |

# A\* Search from Arad to Bucharest

(a) The initial state



# A\* Search from Arad to Bucharest

(a) The initial state



(b) After expanding Arad



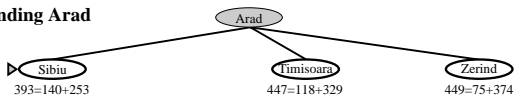
$$f(n) = 140 + 253$$

# A\* Search from Arad to Bucharest

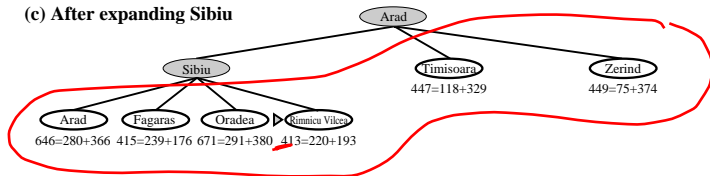
(a) The initial state



(b) After expanding Arad



(c) After expanding Sibiu

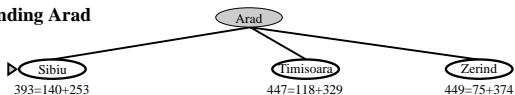


# A\* Search from Arad to Bucharest

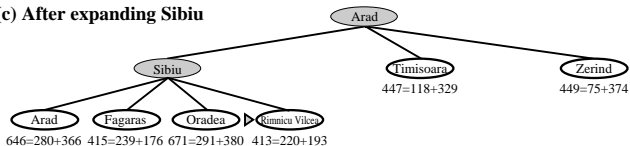
(a) The initial state



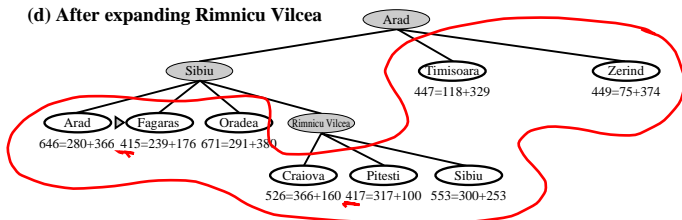
(b) After expanding Arad



(c) After expanding Sibiu

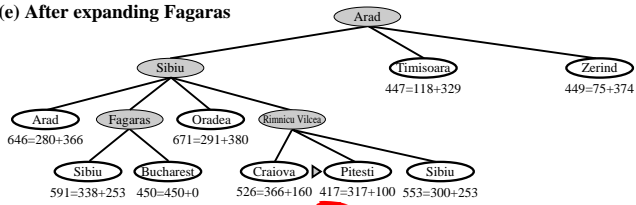


(d) After expanding Rimnicu Vilcea



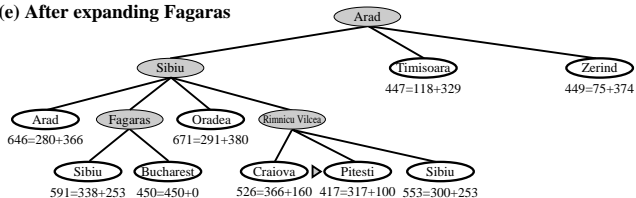
# A\* Search from Arad to Bucharest

(e) After expanding Fagaras

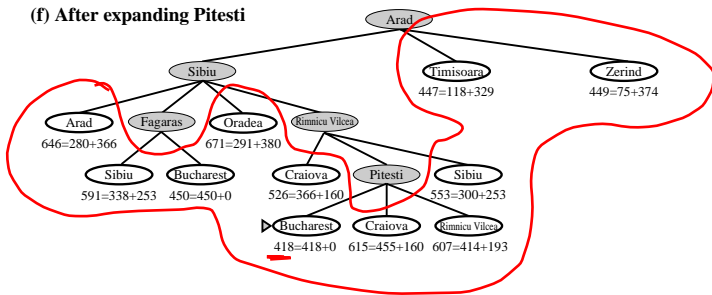


# A\* Search from Arad to Bucharest

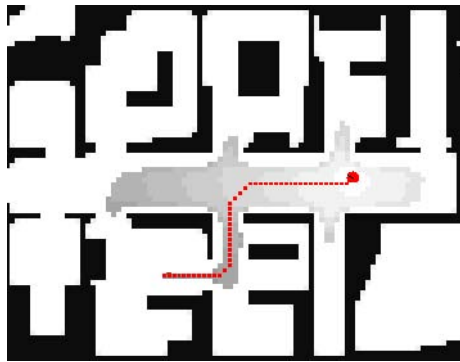
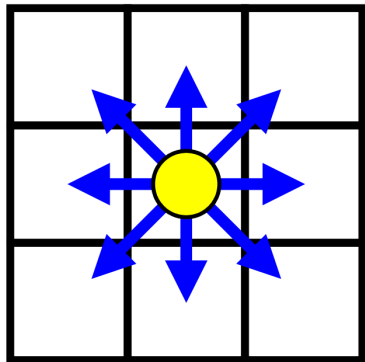
(e) After expanding Fagaras



(f) After expanding Pitesti



# Example: Path Planning for Robots in a Grid-World



Live-Demo: <http://qiao.github.io/PathFinding.js/visual/>

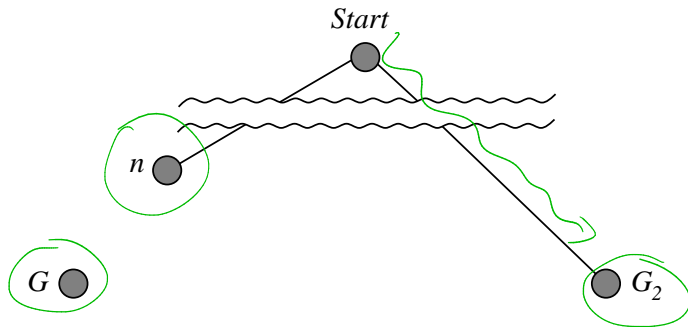


# Optimality of A\*

$$f(n) = g(n) + h(n)$$

**Claim:** The first solution found (= node is expanded and found to be a goal node) has the minimum path cost.

**Proof:** Suppose there exists a goal node  $G$  with optimal path cost  $f^*$ , but A\* has first found another node  $G_2$  with  $\underline{g(G_2)} > \underline{f^*}$ .



# Optimality of $A^*$

Let  $n$  be a node on the path from the start to  $G$  that has not yet been expanded. Since  $h$  is admissible, we have

$$f(n) \leq f^*$$

Since  $n$  was not expanded before  $G_2$ , the following must hold:

$$f(G_2) \leq f(n)$$

and

$$f(G_2) \leq f^*$$

It follows from  $h(G_2) = 0$  that

$$g(G_2) \leq f^*$$

→ Contradicts the assumption!

# Completeness and Complexity

## Completeness:

If a solution exists,  $A^*$  will find it provided that (1) every node has a finite number of successor nodes, and (2) there exists a positive constant  $\delta > 0$  such that every step has at least cost  $\delta$ .

→ there exists only a finite number of nodes  $n$  with  $f(n) \leq f^*$ .

## Complexity:

In general, still exponential in the path length of the solution (space, time)

More refined complexity results depend on the assumptions made, e.g. on the quality of the heuristic function. Example:

In the case in which  $|h^*(n) - h(n)| \leq O(\log(h^*(n)))$ , only one goal state exists, and the search graph is a tree, a sub-exponential number of nodes will be expanded [Gaschnig, 1977, Helmert & Roeger, 2008].

Unfortunately, this almost never holds.

# A note on Graph- vs. Tree-Search

- A\* as described is a tree-search (and may consider duplicates)
- For the graph-based variant, one
  - either needs to consider re-opening nodes from the explored set, when a better estimate becomes known, or
  - one needs to require stronger restrictions on the heuristic estimate: it needs to be **consistent**.

→ A heuristic  $h$  is called **consistent** iff for all actions  $a$  leading from  $s$  to  $s'$ :  $h(s) - h(s') \leq c(a)$ , where  $c(a)$  denotes the cost of action  $a$ .  
(Consistent heuristics prevent the need to re-open nodes from the *explored set*.)

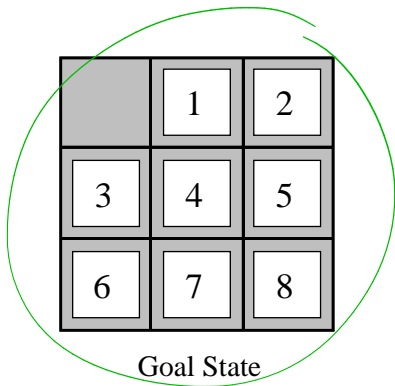
- **Note:** Consistency implies admissibility.
- **Note:** A\* can still be applied if heuristic is not consistent, but optimality is lost in this case.

# Heuristic Function Example

|   |   |   |
|---|---|---|
| 7 | 2 | 4 |
| 5 |   | 6 |
| 8 | 3 | 1 |

Start State

|   |   |   |
|---|---|---|
|   | 1 | 2 |
| 3 | 4 | 5 |
| 6 | 7 | 8 |



Goal State

# Heuristic Function Example

|   |   |   |
|---|---|---|
| 7 | 2 | 4 |
| 5 |   | 6 |
| 8 | 3 | 1 |

Start State

|   |   |   |
|---|---|---|
|   | 1 | 2 |
| 3 | 4 | 5 |
| 6 | 7 | 8 |

Goal State

$h_1$  = the number of tiles in the wrong position

# Heuristic Function Example

|   |   |   |
|---|---|---|
| 7 | 2 | 4 |
| 5 |   | 6 |
| 8 | 3 | 1 |

Start State

|   |   |   |
|---|---|---|
|   | 1 | 2 |
| 3 | 4 | 5 |
| 6 | 7 | 8 |

Goal State

$h_1$  = the number of tiles in the wrong position

$h_2$  = the sum of the distances of the tiles from their goal positions  
(*Manhattan distance*)

# Empirical Evaluation

- $d$  = distance from goal
- Average over 100 instances

Manhattan

| $d$ | Search Cost (nodes generated) |            |            | Effective Branching Factor |            |            |
|-----|-------------------------------|------------|------------|----------------------------|------------|------------|
|     | IDS                           | $A^*(h_1)$ | $A^*(h_2)$ | IDS                        | $A^*(h_1)$ | $A^*(h_2)$ |
| 2   | 10                            | 6          | 6          | 2.45                       | 1.79       | 1.79       |
| 4   | 112                           | 13         | 12         | 2.87                       | 1.48       | 1.45       |
| 6   | 680                           | 20         | 18         | 2.73                       | 1.34       | 1.30       |
| 8   | 6384                          | 39         | 25         | 2.80                       | 1.33       | 1.24       |
| 10  | 47127                         | 93         | 39         | 2.79                       | 1.38       | 1.22       |
| 12  | 3644035                       | 227        | 73         | 2.78                       | 1.42       | 1.24       |
| 14  | -                             | 539        | 113        | -                          | 1.44       | 1.23       |
| 16  | -                             | 1301       | 211        | -                          | 1.45       | 1.25       |
| 18  | -                             | 3056       | 363        | -                          | 1.46       | 1.26       |
| 20  | -                             | 7276       | 676        | -                          | 1.47       | 1.47       |
| 22  | -                             | 18094      | 1219       | -                          | 1.48       | 1.28       |
| 24  | -                             | 39135      | 1641       | -                          | 1.48       | 1.26       |



A\* in general still suffers from exponential memory growth. Therefore, several refinements have been suggested:

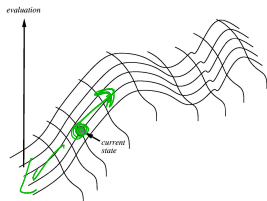
- iterative-deepening A\*, where the f-costs are used to define the cutoff (rather than the depth of the search tree): IDA\*
- Recursive Best First Search (RBFS): introduces a variable  $f\_limit$  to keep track of the best alternative path available from any ancestor of the current node. If current node exceeds this limit, recursion unwinds back to the alternative path.
- other alternatives memory-bounded A\* (MA\*) and simplified MA\* (SMA\*).

# Lecture Overview

- 1 Best-First Search
- 2 A\* and IDA\*
- 3 Local Search Methods**
- 4 Genetic Algorithms

# Local Search Methods

- In many problems, it is unimportant how the goal is reached—only the goal itself matters (8-queens problem, VLSI Layout, TSP).
- If in addition a quality measure for states is given, **local search** can be used to find solutions.
- It operates using a single current node (rather than multiple paths).
- It requires little memory (cp. to most *systematic search strategies*!)
- **Idea**: Begin with a randomly-chosen configuration and improve on it step by step → **Hill Climbing / Gradient Decent**.
- **Note**: It can be used for maximization or minimization respectively (see 8-queens example)



# Example: 8-queens Problem (1)

Example state with heuristic cost estimate  $h = 17$  (counts the number of pairs threatening each other directly or indirectly).

|    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|
| 18 | 12 | 14 | 13 | 13 | 12 | 14 | 14 |
| 14 | 16 | 13 | 15 | 12 | 14 | 12 | 16 |
| 14 | 12 | 18 | 13 | 15 | 12 | 14 | 14 |
| 15 | 14 | 14 | ♠  | 13 | 16 | 13 | 16 |
| ♠  | 14 | 17 | 15 | ♠  | 14 | 16 | 16 |
| 17 | ♠  | 16 | 18 | 15 | ♠  | 15 | ♠  |
| 18 | 14 | ♠  | 15 | 15 | 14 | ♠  | 16 |
| 14 | 14 | 13 | 17 | 12 | 14 | 12 | 18 |

**function** HILL-CLIMBING(*problem*) **returns** a state that is a local maximum

*current*  $\leftarrow$  MAKE-NODE(*problem*.INITIAL-STATE)

**loop do**

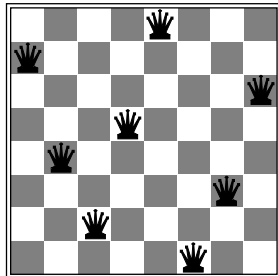
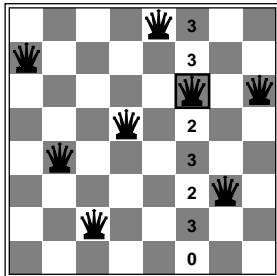
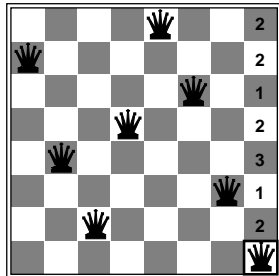
*neighbor*  $\leftarrow$  a highest-valued successor of *current*

**if** *neighbor*.VALUE  $\leq$  *current*.VALUE **then return** *current*.STATE

*current*  $\leftarrow$  *neighbor*

## Example: 8-queens Problem (2)

Possible realization of a hill-climbing algorithm:  
Select a column and move the queen to the square with the fewest conflicts.





- Local maxima: The algorithm finds a sub-optimal solution.
- Plateaus: Here, the algorithm can only explore at random.
- Ridges: Similar to plateaus but might even require suboptimal moves.

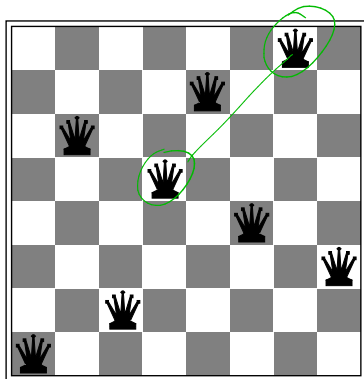
## Solutions:

- *Start over* when no progress is being made.
- “Inject noise” → random walk

Which strategies (with which parameters) are successful (within a problem class) can usually only empirically be determined.

## Example: 8-queens Problem (Local Minimum)

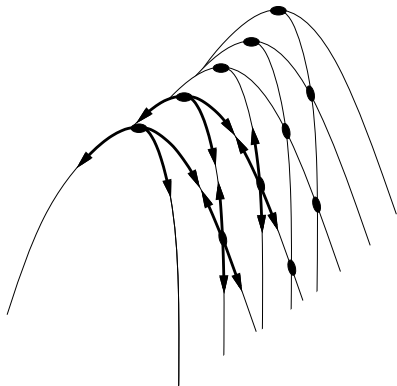
Local minimum ( $h = 1$ ) of the 8-queens Problem. Every successor has a higher cost.





# Illustration of the ridge problem

The grid of states (dark circles) is superimposed on a ridge rising from left to right, creating a sequence of local maxima, that are not directly connected to each other. From each local maximum, all the available actions point downhill.



# Performance figures for the 8-queens Problem

The 8-queens problem has about  $8^8 \approx 17$  million states. Starting from a random initialization, hill-climbing directly finds a solution in about 14% of the cases. On average it requires only 4 steps!

Better algorithm: Allow sideways moves (no improvement), but restrict the number of moves (avoid infinite loops!).

E.g.: max. 100 moves: Solves 94%, number of steps raises to 21 steps for successful instances and 64 for failure cases.

# Simulated Annealing

In the simulated annealing algorithm, “noise” is injected systematically: first a lot, then gradually less.

**function** SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state

**inputs:** *problem*, a problem

*schedule*, a mapping from time to “temperature”

*current*  $\leftarrow$  MAKE-NODE(*problem*.INITIAL-STATE)

**for**  $t = 1$  to  $\infty$  **do**

$T \leftarrow$  *schedule*( $t$ )

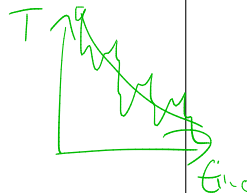
**if**  $T = 0$  **then return** *current*

*next*  $\leftarrow$  a randomly selected successor of *current*

$\Delta E \leftarrow$  *next*.VALUE - *current*.VALUE

**if**  $\Delta E > 0$  **then** *current*  $\leftarrow$  *next*

**else** *current*  $\leftarrow$  *next* only with probability  $e^{\Delta E/T}$



Has been used since the early 80's for VLSI layout and other optimization problems.

- 1 Best-First Search
- 2 A\* and IDA\*
- 3 Local Search Methods
- 4 Genetic Algorithms**

Evolution appears to be very successful at finding good solutions.

*Idea:* Similar to evolution, we search for solutions by three operators: “mutation”, “crossover”, and “selection”.

*Ingredients:*

- Coding of a solution into a string of symbols or bit-string
- A fitness function to judge the worth of configurations
- A population of configurations

*Example:* Represent an individual (one 8-queens configuration) as a concatenation of eight x-y coordinates. Its fitness is judged by the number of non-attacks. The *population* consists of a set of configurations.

# Selection, Mutation, and Crossing

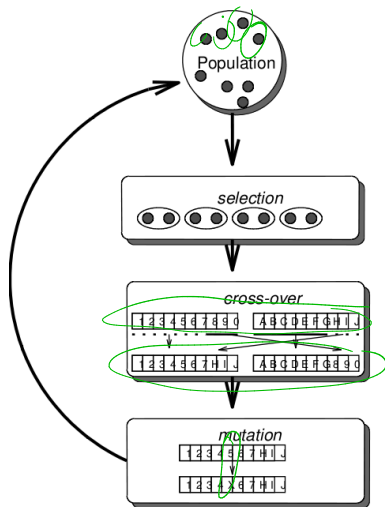
Many variations:

how selection will be applied, what type of cross-over operators will be used, etc.

Selection of individuals according to a fitness function and pairing

Calculation of the breaking points and recombination

According to a given probability elements in the string are modified.



# Summary

- **Heuristics** focus the search
- **Best-first search** expands the node with the highest worth (defined by any measure) first.
- With the minimization of the evaluated costs to the goal  $h$  we obtain a **greedy search**.
- The minimization of  $f(n) = g(n) + h(n)$  combines uniform and greedy searches. When  $h(n)$  is **admissible**, i.e.,  $h^*$  is never overestimated, we obtain the **A\* search**, which is complete and optimal.
- **IDA\*** is a combination of the iterative-deepening and A\* searches.
- **Local search methods** work on a single state only, attempting to improve it step-wise.
- **Genetic algorithms** imitate evolution by combining good solutions.

